Tecnología Informática en un curso de Lenguajes Formales y Teoría de Autómatas: un enfoque constructivista

Carlos I. Chesñevar[1] Ana G. Maguitman[2] María P. González[1] María L. Cobo[1]

[1] **Departamento de Cs. e Ing. de la Computación**
Universidad Nacional del Sur – Av. Alem 1253
B8000CPB Bahía Blanca – Argentina
Tel. 0291-459-5135 / Fax 0291-459-5136
Email: {cic,mpg,lc}@cs.uns.edu.ar

[2] **Computer Science Department**
Indiana University
Bloomington, IN 47405-7104, USA
Email: anmaguit@cs.indiana.edu

Palabras clave: Informática Educativa, teoría de lenguajes formales y autómatas
Remitido a: II Workshop de Tecnología Informática Aplicada a la Educación - IX CACIC

Resumen

El estudio de lenguajes formales y teoría de autómatas (LFTA) juega un rol importante en las carreras de grado de Ciencias de la Computación. La temática involucrada es compleja, dado que contiene muchos conceptos altamente abstractos que requieren un sólido manejo matemático para su comprensión. A partir de experiencias con alumnos de grado de segundo año hemos podido apreciar que muchos estudiantes no se sienten motivados e interesados en ciertos temas fundacionales de la computación en virtud de que los perciben erróneamente como demasiado "orientados hacia las matemáticas".

Este trabajo presenta distintas estrategias didácticas basadas en el uso de tecnología informática que se introdujeron en los últimos semestres de un curso de LFTA para atacar el problema antes planteado. Dichas estrategias estuvieron sustentadas en el marco teórico provisto por el modelo constructivista. Nuestro objetivo principal fue promover un aprendizaje más motivador y significativo de los distintos conceptos teóricos de un curso de LFTA orientado a estudiantes de computación.

1 Introducción y Motivaciones

El estudio de lenguajes formales y teoría de autómatas (LFTA) juega un rol importante en las carreras de grado de Ciencias de la Computación. La temática involucrada es compleja, dado que contiene muchos conceptos altamente abstractos que requieren un sólido manejo matemático para su comprensión. A partir de experiencias con alumnos de grado de segundo año hemos podido apreciar que muchos estudiantes no se sienten motivados e interesados en ciertos temas fundacionales de la computación en virtud de que los perciben como demasiado "orientados hacia las matemáticas". Como paliativo, algunos estudiantes (especialmente aquellos con más dificultades de aprendizaje) tienden a mecanizar la aplicación de propiedades y algoritmos, sin alcanzar un adecuado aprendizaje signficativo.

A fin de que un curso de LFTA resultara más motivador se introdujeron una serie de estrategias didácticas basadas en el uso de tecnología informática, las cuales se integraron con la curricula tradicional. La idea no fue simplificar los contenidos teóricos del curso, sino tornar a éste más

"orientado a la computación", vinculándolo con desarrollo de aplicaciones y otros elementos tecnológicos. Como meta final, se intentó promover un aprendizaje más significativo de los conceptos centrales de LFTA a partir de la introducción de diversas estrategias didácticas basadas en el *modelo constructivista* [Bruner, 1966; Bruner, 1990; Aebli, 1980], entre las que se destacan:

- Lograr una contextualización de la Teoría de la Computación
- Incentivar los procesos de aprendizaje activo y aprendizaje por descubrimiento.
- Aprovechar el conocimiento previo de los estudiantes.
- Aplicar el nuevo conocimiento a través de actividades de afianzamiento y extensión

Este trabajo está estructurado como sigue. Primeramente, en la sección 2 se introduce una breve descripción del modelo constructivista y su relación con la curricula de Ciencias de la Computación. Luego, en la sección 3 se analizan las estrategias antes mencionadas en el contexto de temas centrales de un curso de LFTA. Finalmente, en la sección 4 se presentan las principales conclusiones obtenidas.

2. El Modelo Constructivista y su relación con la curricula de Ciencias de la Computación

El constructivismo es una teoría de aprendizaje que postula que el conocimiento es *construido* de manera activa por el estudiante, y no pasivamente absorbido a partir de clases y libros de texto. Según esta teoría acerca del proceso de enseñanza-aprendizaje la construcción del conocimiento realizada por cada sujeto se modela recursivamente a partir del conocimiento previo del mismo. Las técnicas de enseñanza-aprendizaje derivadas del modelo constructivista involucran explícitamente ese proceso de construcción de conocimiento. Para la teoría constructivista la meta es que el sujeto que aprende logre alcanzar conocimientos significativos, es decir, *modelos mentales* adecuados que estén disponibles para ser utilizados en diferentes contextos.

Existen varias propuestas pedagógicas basadas en el modelo constructivista. Según [Bruner, 1966], el aprendizaje es un proceso activo en el cual los alumnos construyen nuevas ideas o conceptos a partir de conocimientos previos. El alumno selecciona y transforma información, construye hipótesis, toma decisiones, y se sustenta para esto en una *estructura cognitiva* o *modelo mental*, que provee significado y organización a sus experiencias, y le permite ir "mas allá de la información" como simples datos. En este contexto, el profesor debería intentar alentar a los alumnos a descubrir principios por sí mismos. El profesor y el alumno deben involucrarse en un diálogo activo. La tarea del profesor es trasponer didácticamente el conocimiento científico a ser aprendido para adecuarlo al estado actual de entendimiento del alumno. El currículo debería organizarse de forma espiralada, de tal manera que el estudiante continuamente construya nuevos conocimientos sobre aquellos ya aprendidos.

El acercamiento de Bruner se centra en particular en cuatro aspectos centrales: a) la predisposición hacia el aprendizaje, b) las formas en que se puede estructurar un cuerpo de conocimiento para que pueda ser fácilmente captado por el alumno; c) las maneras más efectivas en las que se debe presentar el material a ser aprendido, y d) la disponibilidad de buenos métodos para estructurar conocimiento, que resulten en simplificaciones, generación de nuevas proposiciones y aumento de la capacidad de manipular información por parte del alumno.

En trabajos más recientes [Bruner, 1986 y Bruner, 1990)] Bruner expandió su marco teórico, abarcando también los aspectos *sociales* y *culturales* del aprendizaje. Los principios constructivistas postulados por Bruner pueden sintetizarse como sigue:

1. La enseñanza debe verse involucrada con las experiencias y contextos que hacen que el estudiante esté motivado y sea capaz de aprender.
2. La enseñanza debe ser estructurada de tal forma que pueda ser fácilmente captada por el alumno (organización espiralada).
3. La enseñanza debe ser diseñada para facilitar la adquisición de aprendizaje significativo, permitiendo que el alumno vaya "mas allá" de la información recibida.

El constructivismo ha tenido una influencia significativa en las ciencias de la educación [Glynn, Yeany, & Britton, 1991] y educación matemática [Davis, Maher, & Noddings, 1990]. Como señala [Ben Ari, 2000], citando a [Ernst, 1995, pág. 475] se afirma incluso que *"el constructivismo radical representa el estado del arte en teorías epistemológicas para matemáticas y ciencias de la educación".* No obstante, ha habido mucho menos investigación sobre el constructivismo en la educación en ciencias de la computación. En [Ben Ari, 2000] se indica acertadamente:

> *"El constructivismo ha tenido una extraordinaria influencia en las ciencias en general y en la educación matemática, pero en mucha menor medida en la educación en ciencias de la computación (CSE) […] La literatura sobre constructivismo en CSE no pude compararse de ninguna forma con la vasta literatura existente para la educación en matemática o en física. Aún hoy día, la búsqueda del término* "constructivism" *en la ACM Digital Library retorna solo un puñado de artículos. Si bien muchos educadores en ciencias de la computación se han visto influenciados por el constructivismo, sólo recientemente esto ha sido discutido explícitamente en publicaciones científicas."*

A lo largo de los últimos semestres se han venido integrando distintas estrategias didácticas constructivistas en la curricula de la asignatura "Fundamentos de Ciencias de la Computación", correspondiente al segundo año de las carreras de grado en Ciencias de la Computación que se dictan en el ámbito de la Universidad Nacional del Sur. En distintos casos se aplicó la tecnología informática a través de diferentes alternativas como un recurso pedagógico complementario. El objetivo perseguido fue lograr *aprendizajes significativos* por parte de los alumnos, intentando aumentar su motivación e interés en la asignatura. Dichas estrategias se analizan en detalle en la próxima sección.

3. Estrategias didácticas constructivistas y su relación con el uso de tecnología informática

3.1. Contextualización de Teoría de la Computación en un marco histórico

La inclusión de detalles acerca de la evolución histórica de la teoría de la computación a medida que se iban introduciendo distintos temas teóricos demostró ser de ayuda y de motivación para

nuestros alumnos. Si bien un curso de LFTA no es un curso de historia de la computación, hemos intentado siempre presentar notas biográficas y comentarios vinculados con el contexto histórico en el que emergió la computación como nueva disciplina.

A título de ejemplo, la formalización de la noción de procedimiento efectivo y la presentación de la tesis Turing-Church van acompañadas de algunos hechos relevantes de las vidas de Alonzo Church y Alan Turing. En este contexto los estudiantes descubren que tanto Stephen Kleene como Alan Turing eran dos de los más destacados de los 31 estudiantes que hicieron sus tesis doctorales bajo la guía de Church. De esta manera, se hace más claro por qué las investigaciones de Kleene y Turing se orientan tras objetivos complementarios y muchas veces similares. Nuestra experiencia ha mostrado que conocer este tipo de hechos históricos facilita para los estudiantes la identificación de cómo evolucionaron los conceptos a lo largo del tiempo, y de cómo la investigación científica contribuye a la creación de un nuevo campo del conocimiento, como fue el caso de la teoría de la computación entre 1930 y 1950.

La historia de la teoría de la computación también pone de manifiesto la importancia de la "fertilización cruzada" (cross-breeding) en la investigación, dado que evidencia cómo distintas áreas aparentemente dispares (tales como matemáticas, teoría de lenguajes, e ingeniería) se encontraron en un lugar común a medida que evolucionó las ciencias de la computación. Así, los estudiantes perciben que las ideas de Turing, Kleene y Church en los años 30's, las gramáticas de Chomsky en los años 50's, las redes de Petri y las máquinas de registros de Shepardson Sturgis en los 60's son distintos formalismos que permiten modelar un mismo concepto: *la computabilidad.*

A fin de integrar la historia de la teoría de la computación con los temas teóricos presentados en clase se expandió la página web de la asignatura (http://cs.uns.edu.ar/~cic/fcc.htm) para incluir fotografías y enlaces a distintas notas biográficas y sitios de interés. Nuestra intención fue que aquellos estudiantes que quisieran por caso conseguir el material teórico del curso (por ejemplo, máquinas de Turing) se vieran inducidos también a dar un vistazo a otros sitios asociados al tema (como la tortuosa vida de Alan Turing, y el vínculo entre los desarrollos bélicos en tiempos de guerra y los avances tecnológicos en la computación).

Para facilitar la contextualización de los distintos temas presentados en la asignatura se recurrió también a *mapas conceptuales hipermediales* [Señas y otros, 1996], utilizando para ello transparencias animadas en Microsoft Powerpoint cuya dinámica reflejara la evolución e interconexión entre los distintos formalismos presentados. Estas transparencias animadas fueron utilizadas como recurso pedagógico para la presentación de las clases teóricas de la materia.

3.2. Desarrollo de un aprendizaje activo y por descubrimiento a través del uso de simuladores

En el área de arquitectura de computadoras existen programas educativos específicos denominados *simuladores de computadoras (computer simulators),* que emulan el comportamiento de arquitecturas especializadas o lenguajes ensambladores específicos. Estos programas proveen una excelente herramienta para posibilitar el aprendizaje activo de conocimiento especializado utilizando abstracción, interacción y visualización.

En el área de LFTA se ha extendido recientemente la noción anterior a los denominados "simuladores de computadora teóricos" (*theoretical computer simulators*), que posibilitan simular autómatas de estado finito, máquinas de Turing y otros varios modelos de autómatas. En la asignatura "Fundamentos de Ciencias de la Computación" hemos integrado diversos simuladores de computadora teóricos, complementando los distintos modelos formales de computación presentados en la materia. En trabajos recientes [Chesñevar y otros; 2003; Chesñevar y Cobo, 2002] se analizaron las características principales de varias herramientas educativas para la enseñanza de LFTA, distinguiéndose dos categorías principales, a saber:

1) Paquetes de software genéricos multi-propósito para enseñar e integrar distintos conceptos interrelacionados de LFTA. Se destacan aquí JFLAP [Rodger y otros, 1996], DEM (desarrollado a partir de los formalismos presentados en [Taylor, 1998]) y Minerva [Estrebou y otros, 2002]).
2) Herramientas de software orientadas hacia simular una clase específica de autómatas con propósitos educativos.

Los simuladores de computadora teóricos proveen un interesante enlace entre la teoría y la práctica. En nuestra opinión, este tipo de entornos refuerza la importancia y el significado de muchas cuestiones teóricas que son relevantes al momento de abordar problemas prácticos. Según nuestra experiencia personal, la mayoría de los tópicos en un curso tradicional de LFTA pueden ilustrarse con este tipo de simuladores, los cuales pueden conseguirse generalmente como software de libre distribución vía Internet. Los programas multipropósito (como **Minerva** [Estrebou y otros, 2002] **y JFLAP** [Hung y Rodger, 1997]) son una buena alternativa, en el sentido en que proveen una visión unificada de distintos tipos de autómata. No obstante, nuestra experiencia docente mostró que los alumnos prefieren trabajar con varios simuladores alternativos para el mismo autómata. Debe acentuarse, sin embargo, que tales simuladores son sólo una herramienta y no el tema central del curso. El uso de simuladores como ayudas para la enseñanza ha resultado sumamente provechoso, alentando a un aprendizaje activo y por descubrimiento por parte de los alumnos. Para un análisis más detallado puede consultarse [Chesñevar y otros, 2003].

3.3. Uso de conocimientos previos para vincular teoría de la computación con lenguajes de programación.

La mayoría de los libros de texto de LFTA [Lewis y Papadimitrou, 1998; Hopcroft y Ullman, 1972; Taylor, 1998] enfatizan la importancia de un buen entendimiento de las nociones abstractas de computabilidad, lenguajes recursivos vs. no recursivos, etc. No obstante, es muy poco común encontrar referencias concretas o ejemplos que vinculen conceptos de LFTA con lenguajes de programación concretos conocidos por los alumnos. Entendemos que un lenguaje de programación constituye en sí mismo una herramienta tecnológica que facilita el aprendizaje significativo de varios conceptos teóricos abstractos. En nuestro caso particular, en la asignatura "Fundamentos de Cs. De la Computación" hemos introducido varios ejemplos que vinculan el lenguaje Pascal (ya conocido en profundidad por los alumnos en ese punto de la carrera) con algunos conceptos de LFTA. Seguidamente se indican algunos ejemplos.

Ejemplo 1: Considérese la definición de lenguaje recursivo:

> **Def**.: Dado un alfabeto Σ, un lenguaje L es *recursivo* en Σ sssi para alguna cadena w ∈ Σ* existe un procedimiento efectivo para decidir si w∈L o bien w∉L.

En lugar de utilizar un lenguaje formal abstracto para ilustrar este concepto, sugerimos a nuestros alumnos que consideren primero el lenguaje **ValidPrograms**={w | w es un programa en Pascal válido }, definido sobre el alfabeto Σ de todos los caracteres ASCII. En otras palabras, **ValidPrograms** es el conjunto (potencialmente infinito) de todas aquellas cadenas {w_1,w_2, …, w_k, ….} tales que cada w_i es el código de un programa Pascal que puede compilarse y ejecutarse exitosamente. Los estudiantes luego son confrontados con la pregunta *"**ValidPrograms** ¿es un lenguaje recursivo?"*. La respuesta es sí, y el razonamiento que la justifica es simple: si este no fuera el caso, el compilador de Pascal no sería capaz de discriminar, para un archivo de texto particular T, si el mismo es un programa Pascal válido o no.

Ejemplo 2: Siguiendo la misma línea de razonamiento, consideramos también la propiedad de que los lenguajes recursivos son cerrados bajo complemento. Esto puede probarse formalmente como se hace en [Lewis y Papadimitrou, 1998]. Sin embargo, entendemos que es útil motivar primero a los alumnos con un lenguaje recursivo concreto (como el dado en el ejemplo 1) y luego plantear la siguiente pregunta: "*¿Es posible diseñar un compilador "anti-Pascal", esto es un compilador que rechace todo programa correcto y acepte todo aquel que no lo sea?"*. Para muchos estudiantes la respuesta es afirmativa, y el razonamiento es inmediato: basta cambiar el mensaje en la pantalla, reemplazando el mensaje "Compilación exitosa" por "Error en compilación", y viceversa. Esta idea es justamente la que se utiliza para demostrar el teorema que formalmente establece que si L es un lenguaje recursivo, entonces su complemento también lo es. Después de ver este ejemplo, la prueba formal asociada presentada luego resulta mucho más natural y comprensible para los alumnos.

Ejemplo 3: Considérese nuevamente el lenguaje **ValidPrograms** ={w | w es un programa válido en Pascal}. ¿Es este lenguaje libre de contexto? A través de algunos ejemplos, inducimos a los estudiantes a pensar acerca del hecho de que todo identificado de Pascal tiene que ser declarado antes de ser usado. Esta característica sugiere intuitivamente que el conjunto de todas las cadenas de **ValidPrograms** no puede capturarse con una gramática libre de contexto. Posteriormente, se demuestra formalmente que **ValidPrograms** no es libre de contexto utilizando propiedades de clausura y homomorfismos para lenguajes libres de contexto.

La vinculación entre tópicos de LFTA con un lenguaje de programación concreto (en nuestro caso con Pascal, que es el primer lenguaje de programación utilizado por los alumnos) ha demostrado ser muy motivador para introducir muchos conceptos abstractos de teoría de lenguajes formales. Si bien el curso de LFTA no aborda los temas propios del diseño y construcción de compiladores (hay una asignatura más avanzada dedicada enteramente a este tema), muchas ideas relacionadas con aspectos léxicos, sintácticos y semánticos de los lenguajes de programación aparecen en discusiones, preguntas y diálogos con los estudiantes.

Al analizar los alcances y limitaciones de distintos tipos de gramáticas formales, es útil resaltar las características de Pascal que pueden capturarse a través de distintos lenguajes en la jerarquía de Chomsky. Así, por caso, los estudiantes reconocen que los diagramas de Backus-Naur (usados comúnmente para definir la sintaxis de Pascal) son equivalentes a las gramáticas libres de contexto (tipo 2 en la jerarquía de Chomsky). También resulta evidente que no todo árbol de derivación que

describe un programa sintácticamente válido corresponde a un programa semánticamente válido. En teoría, los aspectos semánticos de Pascal podrían especificarse utilizando una gramática sensible al contexto (tipo 1 en la jerarquía de Chomsky), pero tal tipo de acercamiento sería excesivamente complejo en la práctica. En consecuencia, tales aspectos se especifican usualmente en lenguaje natural a través de sentencias de la forma "los identificadores deben declararse antes de ser usados" o bien "los parámetros efectivos deben concordar en cantidad y tipo con los parámetros formales en una invocación", etc.

Las relaciones anteriores resultan esclarecedoras para los estudiantes, ya que éstos pueden ver ejemplos representativos que que ilustran la correspondencia entre cuestiones semánticas y lenguajes abstractos estudiados en clase. Continuando con la relación entre aspectos de Pascal y la jerarquía de Chomsky, también se introduce el hecho de que la potencia computacional de Pascal es equivalente a la de una máquina de Turing, por lo que los programas en Pascal son tan potentes como una gramática estructurada por frases (tipo 0 en la jerarquía de Chomsky). A partir de los conceptos enunciados se plantean luego distintas charlas con los alumnos, que llevan a las siguientes conclusiones:

- Al analizar un lenguaje dado por un conjunto de *tokens* (como los que se utilizan en Pascal) se requiere la expresividad de lenguajes regulares. Así, la definición de operadores, delimitadores, números válidos e identificadores puede representarse a través de expresiones regulares. Un tratamiento en profundidad del tema se continúa luego en el desarrollo de un analizador léxico, en un curso de Construcción de Compiladores.

- Al analizar un lenguaje dado por un conjunto de cadenas que corresponde a programas sintácticamente válidos, se requiere la potencia de los lenguajes libres de contexto, dado que muchos elementos del lenguaje de programación trabajan "de a pares" (ej. Las palabras reservadas BEGIN y END como delimitadores). El Pumping Lemma para Lenguajes Regulares [Lewis y Papadimitrou, 1998] puede usarse para mostrar que ciertas sentencias estructuradas no corresponden a un lenguaje regular. Por caso, el lenguaje dado únicamente por sentencias "if-then-else" anidadas es no regular, pero puede ser especificado con una gramática libre de contexto. Una vez más, un tratamiento detallado del tema está fuera del alcance del curso y se posterga hasta el desarrollo de un analizador sintáctico en un curso de Construcción de Compiladores.

- Al analizar ciertas cuestiones semánticas se requiere el poder de los lenguajes sensibles al contexto. En un programa Pascal, un identificador solo puede usarse si ha sido declarado previamente. Análogamente, los tipos no predefinidos tienen que ser declarados antes de ser usados en una declaración, y las funciones y procedimientos no pueden usarse si no han sido declarados previamente. El lenguaje $L=\{ww \mid w \text{ está en } \{a,b\}^*\}$ puede verse como una abstracción de las situaciones mencionadas. Este es un caso prototípico de un lenguaje sensible al contexto, pero no libre de contexto. En un curso posterior de Construcción de Compiladores se introducen para esto las gramáticas de atributos o los modelos basados en tablas para especificar los requerimientos sensibles al contexto de un lenguaje de programación, implementándoselos luego a través de un Analizador Semántico.

En todos los casos citados se utilizó el lenguaje Pascal como una herramienta tecnológica que posibilita establecer conexiones entre la teoría de lenguajes abstractos y sus implicancias y valor práctico en el ámbito de un lenguaje de programación conocido por los alumnos.

3.4. Artículos recientes sobre teoría de la computación como actividad de refuerzo y extensión

Siguiendo a Hans Aebli [Aebli, 1980], un aspecto importante en el proceso de enseñanza bajo el enfoque constructivista involucra la "aplicación significativa". Las actividades de afianzamiento y extensión propuestas por este enfoque buscan que el aprendizaje se refuerce adquiriendo funcionalidad y operatividad. El término "extensión" se refiere al proceso que vincula el conocimiento adquirido con nuevas situaciones problemáticas, ya sea dentro del mismo área de conocimiento o otras áreas de conocimiento, incluyendo la vida cotidiana. La aplicación concreta de nuevos aprendizajes debería estar –según Aebli- tanto al comienzo como a la finalización de todo proceso de aprendizaje. A fin de concretizar actividades de afianzamiento y extensión en un curso de LFTA, se introdujeron ciertos artículos y desarrollos recientes que se vinculan con la teoría de la computación.

Los últimos semestres presentamos distintos artículos técnicos como material complementario a la asignatura, los que encontraron una buena aceptación entre muchos estudiantes. En nuestra opinión, esto se debió en parte al hecho de los alumnos percibieron que eran capaces de comprender y discutir un texto técnico destinado a profesionales en Ciencias de la Computación. Seguidamente se sintetizan las características de los principales artículos utilizados:

- En [Teuscher, 2002] se analiza la posibilidad de implementar una "hipercomputadora", una super máquina de Turing capaz de ir más allá de los límites de la computabilidad convencional. Alan Turing mismo presentó un modelo de hipercomputadora (la "máquina-O", una máquina universal de Turing aumentada con un oráculo que lleva a cabo una computación que la Máquina Universal no puede computar en tiempo finito). Como se indica en [Teuscher, 2002], Turing no dio indicación de cómo implementar este tipo de oráculo. Si bien estas hipercomputadoras son aún hoy dispositivos puramente teóricos, en este ámbito aparecen varias cuestiones abiertas, principalmente en el área de la Inteligencia Artificial. A través de este artículo, los estudiantes no solo confirman que la importancia de las ideas de Turing como punto de referencia para la comunidad científica en Ciencias de la Computación, sino que también vislumbran las especulaciones más recientes en torno a los límites de lo que las computadoras pueden hacer.
- En [Burgin, 2001] se aborda la temática de qué es capaz de hacer la tecnología dentro de los límites de la computabilidad. Este artículo resultó particularmente interesante, en virtud de que acentúa la importancia de una buena teoría formal en Computación como base para el desarrollo de aplicaciones prácticas futuras. En este artículo, Mark Burgin incluye una pequeña anécdota acerca de la primer computadora electrónica creada en los Estados Unidos. El matemático John Von Neumann había sido invitado a estudiarla y analizarla. Burgin señala:

  > *``Cuando le explicaron a Von Neumann los principios del funcionamiento de esta computadora electrónica, él sugirió una arquitectura de computadora más avanzada (hoy conocida como ``arquitectura de Von Neumann"). Durante largo tiempo, todas las computadoras tuvieron la arquitectura de Von Neumann, pese a que muchos otros componentes electrónicos [...] cambiaron muy rápidamente. Sin embargo, pocos saben que esta arquitectura copió en parte la estructura de una máquina de Turing. Von Neumann mismo no explicó esto,*

pero dado que era un experto en teoría de algoritmos, conocía en detalle [el poder de] las máquinas de Turing."

El análisis de Burgin nos ayudó a enfatizar a nuestros alumnos que un buen entendimiento de las bases teóricas de la computación resulta fundamental para el desarrollo de aplicaciones prácticas.

- Las consideraciones teóricas para el desarrollo de antivirus resultó también interesante como actividad de extensión para los alumnos. En [Nachenberg, 1997], el autor –Investigador Principal en la compañía Symantec- describe muchas técnicas utilizadas en la guerra entre virus y antivirus. El autor hace referencia a la imposibilidad de crear un programa antivirus invulnerable. Aún las tecnologías de antivirus más sofisticadas pueden ser blanco de ciertos tipos de virus polimórficos. La prueba de esta afirmación está ligada estrechamente a la demostración del famoso "halting problem" para máquinas de Turing.

- Finalmente, y ya en un plano más filosófico, el conocido artículo de Peter Wegner "*Why Interaction Is More Powerful Than Algorithms*" [Wegner, 1997] resultó también fuente de discusión entre los alumnos. En este artículo, el autor argumenta que la incorporación de interacción da lugar a modelos nuevos de computación que son irreducibles al paradigma algorítmico. Dado que muchos estudiantes ya han trabajado con ciertos lenguajes de programación que incorporan la posibilidad de utilizar la interacción para modelar ciertos problemas (como por ejemplo con el disparo de eventos a través de pulsaciones del mouse), la pregunta natural que surge es si tales lenguajes superan los límites de la computabilidad impuestos por las máquinas de Turing.

4. Conclusiones

La enseñanza de la teoría de la computación es una tarea motivadora y llena de desafíos, en la que los estudiantes deberán confrontar preguntas fundamentales, tales como *"¿qué tipos de cosas pueden computarse? "* o "*¿cuáles son los límites de la computabilidad?*". Es claro que una buena base matemática es un requisito insoslayable para poder comprender claramente la respuesta a tales preguntas.

Sin embargo, como se ha detallado a lo largo de este trabajo, nuestra experiencia ha mostrado que muchos estudiantes de computación tienden a ver ciertos tópicos de LFTA como demasiado "orientados a las matemáticas". Esto lleva a cierta falta de motivación e interés de su parte, lo que incide negativamente en su desempeño en la materia. Nuestra propuesta no consiste en adoptar una visión informal de estos temas, simplificando la teoría subyacente, sino en introducir diferentes estrategias didácticas que hagan que estos temas resulten más interesantes y motivadores para los estudiantes.

Fue con el objetivo antes descripto que gradualmente fuimos incorporando distintos elementos de tecnología informática en dos planos: como herramienta pedagógica para el profesor (por caso en el uso de los simuladores descriptos en la sección 3.2) y como elementos motivadores adicionales para los alumnos (como en el caso de los artículos y notas vinculadas a tecnología y su relación con la teoría de la computación mencionados en la sección 3.4). Esta integración se realizó bajo un

enfoque constructivista, de manera tal que la tecnología contribuyera de manera positiva en la concreción de un aprendizaje significativo de los conceptos presentados.

Las estrategias delineadas en este trabajo fueron aplicadas en los últimos tres semestres en un curso cuatrimestral denominado "Fundamentos de Ciencias de la Computación" (Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina). Distintos relevamientos efectuados entre los alumnos (incluyendo encuestas y charlas personalizadas) evidenciaron resultados positivos: muchos de los tópicos "difíciles" y tradicionalmente "aburridos" de la materia (tales como la teoría de lenguajes recursivos, que en un principio no parecía despertar mucho interés en el alumnado) resultaron más interesantes y generadores de discusiones e intercambios de ideas entre los alumnos.

En nuestra opinión, las estrategias didácticas presentadas en este trabajo constituyen un excelente complemento que enriquece la curricula tradicional de LFTA, tornándola más significativa y motivadora para los estudiantes de carreras de computación. Como resultado final, los estudiantes logran una mejor apreciación del rol de la teoría de la computación, no solo en el contexto puramente técnico de la disciplina sino también en relación con su rol y trascendencia con relación a la sociedad.

Bibliografía

Aebli, H. (1980) - *Denken: Das Ordnen des Tuns – Kognitive Aspekte der Handlungstheorie /Denkprozesse. Stutgart, Klett-Cotta.*

Augusto, J.C (1995) *Fundamentos de Ciencias de la Computación – Notas de Curso*. Universidad Nacional del Sur, Argentina.

Bilska, A.O. et al. (1997) A Collection of Tools for Making Automata Theory and Formal Languages Come Alive. *ACM SIGCSE Bulletin 29,* 1, 15-19.

Bruner, J. (1966). Toward a Theory of Instruction. Cambridge, MA: Harvard Univ. Press, 1966.

Bruner, J. (1986) Actual Minds, Possible Worlds. Cambridge, MA – Harvard Univ. Press, 1986.

Bruner, J. (1990) Acts of Meaning. Cambridge, MA – Harvard Univ. Press, 1990.

Burgin, M. (2001) How We Know What Technology Can Do - *CACM*, Nov. 2001, Vol.44, No. 11, pages 83-88.

Chesñevar, C. I., Cobo, M. L., and Yurcik, W (2003).. *Using Theoretical Computer Simulators for Formal Languages and Automata Theory*. ACM SIGCSE Bulletin, Vol. 35, No. 2.

Chesñevar, C. I, Cobo, M.L (2002)- *Simulators for Teaching Formal Languages and Automata Theory: A comparative Survey*. En Procs. del VIII Congreso Argentino en Ciencias de la Computación, págs. 1089-1097. Buenos Aires, Argentina.

Estrebou, F. et al. (2002) Minerva: Una Herramienta Para un Curso de Lenguajes Formales y Autómatas. *Latinamerican Conference in Informatics (CLEI),* Montevideo, Uruguay.

Gramond, E. and S.H. Rodger. (1999) Using JFLAP to Interact with Theorems in Automata Theory. *ACM SIGCSE Bulletin 31*, 1, 336-340.

Grinder, M.T. (2002) Animating Automata: A Cross-Platform Program for Teaching Finite Automata. *ACM SIGCSE Bulletin 34*, 1, 63-67.

Grinder, M.T. et al. (2002) Loving to Learn Theory: Active Learning Modules for the Theory of Computing. *ACM SIGCSE Bulletin 34*, 1, 371-375.

Hopcroft, J. and Ullman, J. (1979) *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, (1979).

Hung, T. and Roger, S. H. (2000) Increasing Visualization and Interaction in the Automata Theory Course. *ACM SIGCSE Bulletin 32*, 1 (2000), 6-10.

Ben-Ari, Mordechai (2000). Constructivism in Computer Science Education – *Journal of Computers in Mathematics and Science Teaching*.

Nachenberg, C. (1997). Computer Virus-Antivirus Coevolution. - *CACM*, Vol. 40, No. 1, pages 46-51.

Lewis, H. and Papadimitriou, C. (1998) *Elements of the Theory of Computation 2nd Edition.* Prentice-Hall.

McDonald, J. (2002) Interactive Pushdown Automata Animation. *ACM SIGCSE Bulletin 34*, 1, 376-380.

Robinson, M.B. et al. (1999) A Java-based Tool for Reasoning About Models of Computation Through Simulating Finite Automata and Turing Machines. *ACM SIGCSE Conference*, 105-109.

Señas, P.; Moroni, N.; Vitturini, M.; Zanconi, M. (1996) – *"Combining Concept Mapping and Hypermedia" – en Ed-Media 96, Boston, USA.*

Taylor, G. (1998) *Models of Computation and Formal Languages*. Oxford University Press.

Teuscher, C. and Sipper, M., (2002) Hyptercomputation: Hype or Computation? - *CACM*, Vol.45, No. 8, pages 23-24.

Wegner, P. (1997). Why Interaction is More Powerful Than Algorithms. CACM, Vol 40, No. 5, pages 80-91.